UNIFESP
UNIVERSIDADE FEDERAL DE SÃO PAULO
1933
Instituto de Ciência e Tecnologia
Universidade Federal de São Paulo

Compiladores:
Análise Léxica – De AFND para AFD e sua implementação

Profª Thaína A. A. Tosta

tosta.thaina@unifesp.br

São José dos Campos – 2021/2

De AFND para AFD e sua implementação

- Um programa que implementa um AFD é mais eficiente no reconhecimento de cadeias do que um programa que implementa um AFND;
- Por esse motivo, é vantajoso encontrar o AFD equivalente ao AFND;
- Construção de Subconjuntos: algoritmo para a construção de um AFD a partir de um AFND onde cada estado do AFD construído corresponde a um conjunto de estados do AFND.

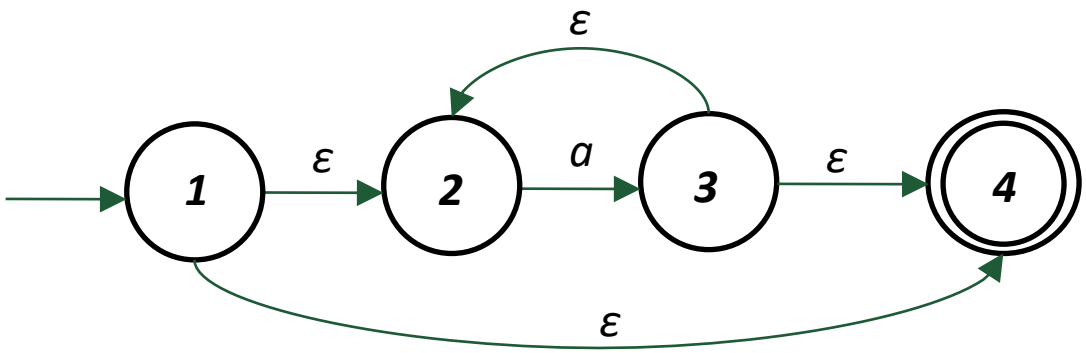
De AFND para AFD e sua implementação

- O algoritmo de construção de subconjuntos requer a eliminação das *ε-transições* do AFND;
- A eliminação das *ε-transições* requer a construção de *ε-fechos;*
- *ε-fecho* de um estado *s* é o conjunto de estados atingíveis por uma série de zero ou mais *ε-transições*
  - Denotamos esse conjunto como $\bar{s}$
  - O *ε-fecho* de um estado sempre contém o próprio estado

De AFND para AFD e sua implementação

ε-fecho - Exemplo

Considere o AFND correspondente à expressão regular a*



$\overline{1} = \{1, 2, 4\}$

$\overline{2} = \{2\}$

$\overline{3} = \{2, 3, 4\}$

$\overline{4} = \{4\}$

De AFND para AFD e sua implementação

- Definimos o ε-fecho de um conjunto de estados como a união dos ε-fechos de cada estado individual

$$\overline{S} = \bigcup_{s \text{ em } S} \overline{s}$$

- Exemplo

  Considere o AFND da expressão regular a*

  $\overline{1} = \{1, 2, 4\}$ e $\overline{3} = \{2, 3, 4\}$

  $\overline{\{1, 3\}} = \overline{1} \cup \overline{3} = \{1, 2, 3, 4\}$

De AFND para AFD e sua implementação

Construção de Subconjuntos

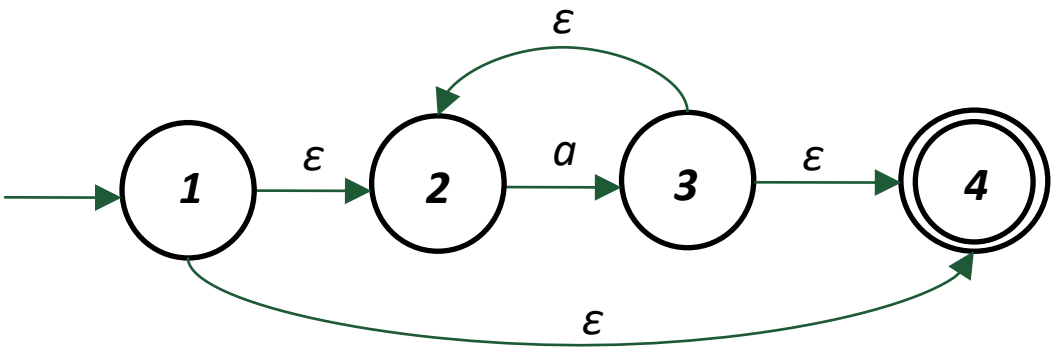
Denominaremos de $\overline{M}$ o AFD construído a partir do AFND M

- 1º passo: computamos o ε-fecho do estado inicial de M, que passa a ser o estado inicial de $\overline{M}$, resultando no conjunto S
- 2º passo: para o conjunto S, e para cada conjunto subsequente, computamos transições de caracteres *a,* que denotamos da seguinte forma:

  S'_a = {t | para algum *s* em *S* existe uma transição de *s* para *t* em *a*}
- 3º passo: computamos $\overline{S'_a}$ (ε-fecho de S'_a)
  - Isso define um novo estado na construção de subconjuntos, juntamente com uma nova transição $S \xrightarrow{a} \overline{S'_a}$
  - Aplicamos os passos 2 e 3 no conjunto resultante de $\overline{S'_a}$ e assim sucessivamente até que novos estados e transições não sejam mais criados
- 4º passo: marcamos como estados de aceitação de $\overline{M}$ os subconjuntos que contenham estados de aceitação de M.

De AFND para AFD e sua implementação

Construção de Subconjuntos



1º passo: $\overline{\{1\}} = \{1, 2, 4\}$

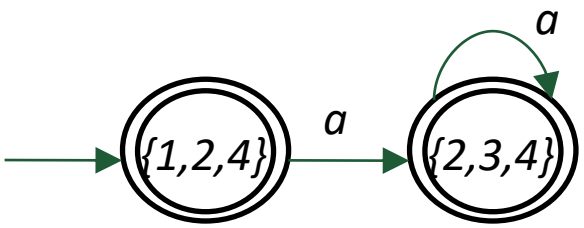
2º passo: $\{1, 2, 4\}_a = \{3\}$

3º passo: $\overline{\{3\}} = \{2, 3, 4\}$

2º passo: $\{2, 3, 4\}_a = \{3\}$

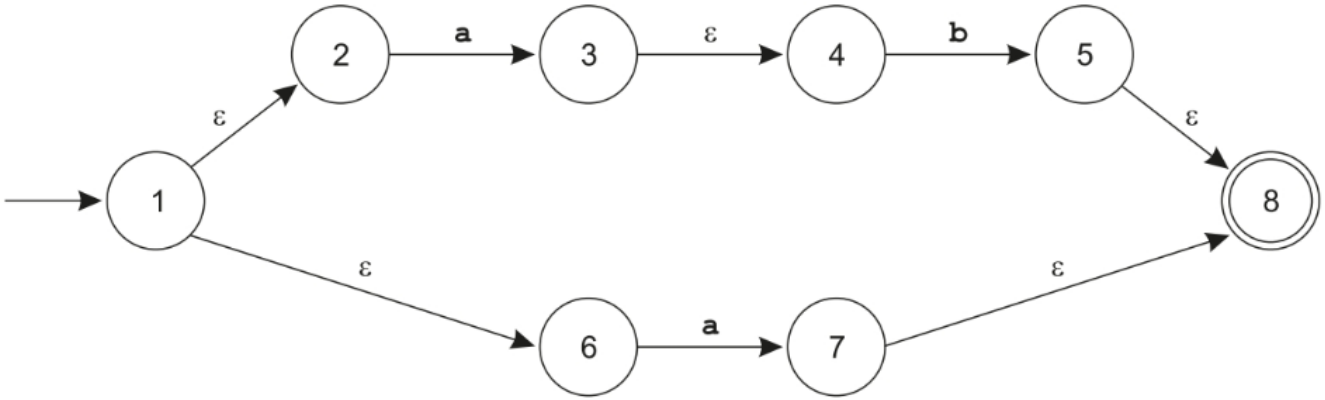
3º passo: $\overline{\{3\}} = \{2, 3, 4\}$

4º passo: Estados finais que tenham o estado 4



De AFND para AFD e sua implementação

Construção de Subconjuntos



1º passo: $\overline{\{1\}} = \{1, 2, 6\}$

2º passo: $\{1, 2, 6\}_a = \{3, 7\}$

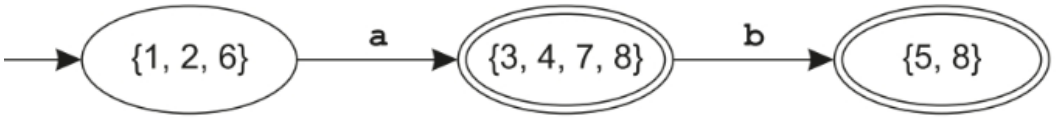
3º passo: $\overline{\{3, 7\}} = \{3, 4, 7, 8\}$

2º passo: $\{3, 4, 7, 8\}_b = \{5\}$

3º passo: $\overline{\{5\}} = \{5, 8\}$

4º passo: Estados finais que tenham o estado 8

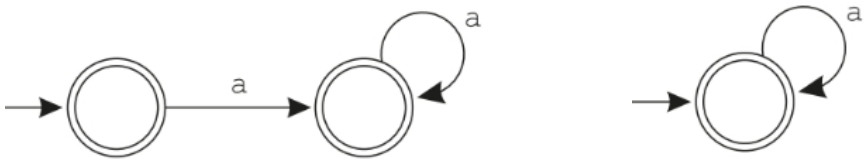
a|ab



De AFND para AFD e sua implementação

Minimização de estados em um AFD

Os algoritmos apresentados para construir um AFD a partir de uma expressão regular não garantem um AFD com o menor número de estados;



É importante encontrarmos um AFD ótimo, ou seja, com o número mínimo de estados;

Pela teoria de autômatos, dado um AFD, existe um AFD equivalente com um número mínimo de estados, o qual é único.

De AFND para AFD e sua implementação

Minimização de estados em um AFD

- Método da construção de subconjuntos gera AFDs, possivelmente, com estados redundantes;

- O procedimento de minimização permite obter autômatos equivalentes baseado no particionamento sucessivo do conjunto de estados.

De AFND para AFD e sua implementação

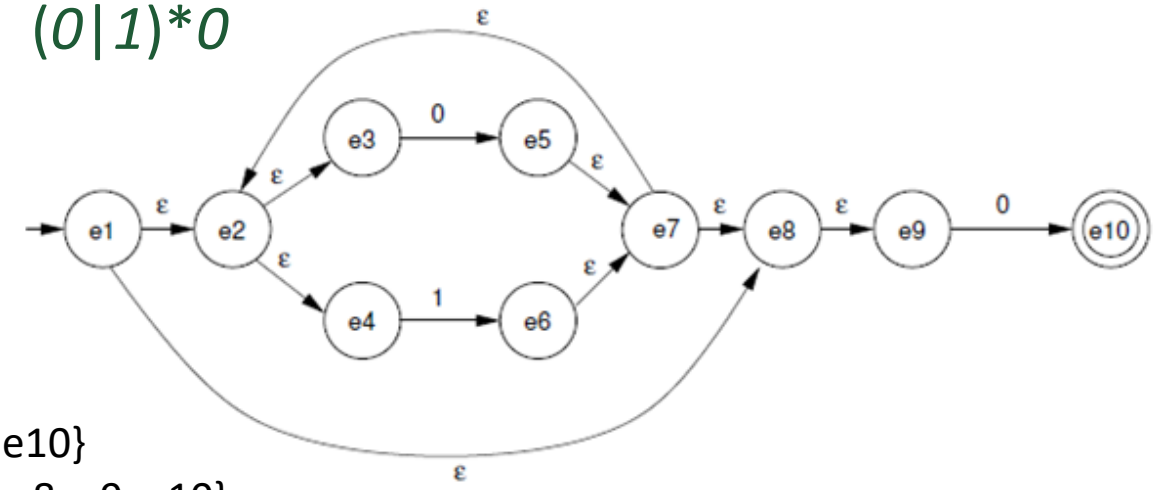
Minimização de estados em um AFD

- Particionar os estados do AFD (inicialmente em dois conjuntos):
  - C_1 = {todos estados de aceitação}
  - C_2 = {todos estados que não são de aceitação}
- Avaliar as transições de estados em cada conjunto
  - Se as transições levarem para conjuntos de estados idênticos, os estados analisados são redundantes
- Combinar estados redundantes (se identificados)

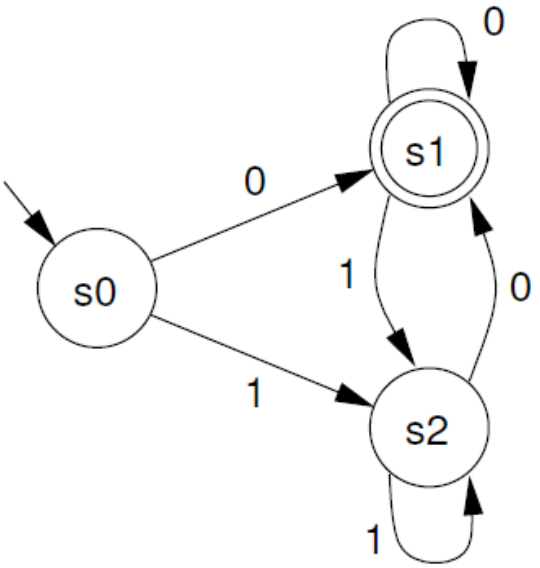
De AFND para AFD e sua implementação

Atentem-se à construção dos ε-fecho, como $\overline{e5}$ = {e2, e3, e4, e5, e7, e8, e9} (***Todos*** os estados alcançados com ε-transições)

*(0|1)*0*



1ª passo: $\overline{e1}$ = {e1, e2, e3, e4, e8, e9}
2º passo: {e1, e2, e3, e4, e8, e9}$_0$ = {e5, e10}
3º passo: $\overline{\{e5, e10\}}$ = {e2, e3, e4, e5, e7, e8, e9, e10}
2º passo: {e1, e2, e3, e4, e8, e9}$_1$ = {e6}
3º passo: $\overline{\{e6\}}$ = {e2, e3, e4, e6, e7, e8, e9}
2º passo: {e2, e3, e4, e5, e7, e8, e9, e10}$_0$ = {e5, e10}
3º passo: $\overline{\{e5, e10\}}$ = {e2, e3, e4, e5, e7, e8, e9, e10}
2º passo: {e2, e3, e4, e5, e7, e8, e9, e10}$_1$ = {e6}
3º passo: $\overline{\{e6\}}$ = {e2, e3, e4, e6, e7, e8, e9}
2º passo: {e2, e3, e4, e6, e7, e8, e9}$_0$ = {e5, e10}
3º passo: $\overline{\{e5, e10\}}$ = {e2, e3, e4, e5, e7, e8, e9, e10}
2º passo: {e2, e3, e4, e6, e7, e8, e9}$_1$ = {e6}
3º passo: $\overline{\{e6\}}$ = {e2, e3, e4, e6, e7, e8, e9}
4º passo: {e2, e3, e4, e5, e7, e8, e9, e10} é estado final



De AFND para AFD e sua implementação

Minimização de estados em um AFD

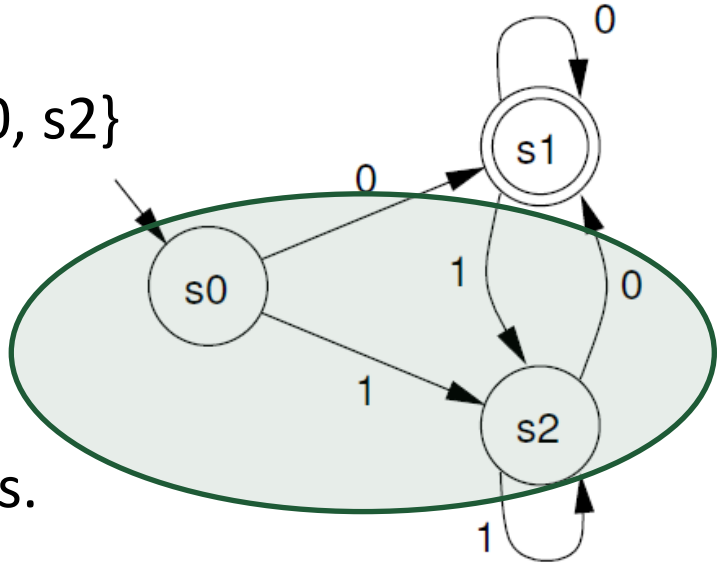
Para o autômato obtido para a expressão *(0|1)*0*:

1. Partição inicial $P_1 = \{C_1, C_2\}$, com

$$C_1 = \{s1\} \quad \text{e} \quad C_2 = \{s0, s2\}$$

2. Para a partição C_2:

	s0	s2
0	C_1	C_1
1	C_2	C_2

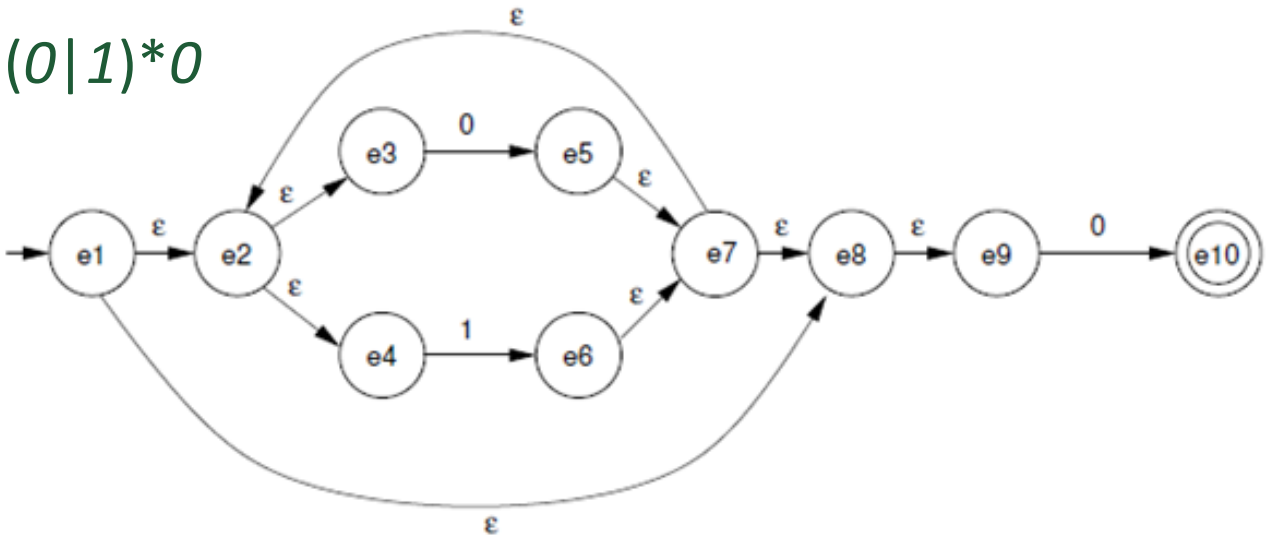


Assim, os estados s0 e s2 são redundantes.

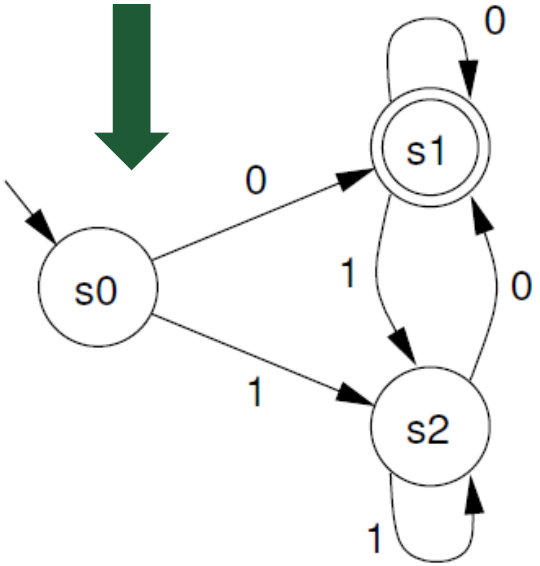
Para uma discussão mais detalhada sobre o tema de minimização de estados de um AFD, **ver Aho *et al.* Seção 3.9.6**

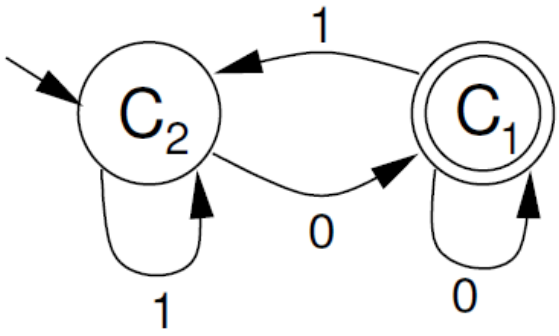
De AFND para AFD e sua implementação

*(0|1)*0*



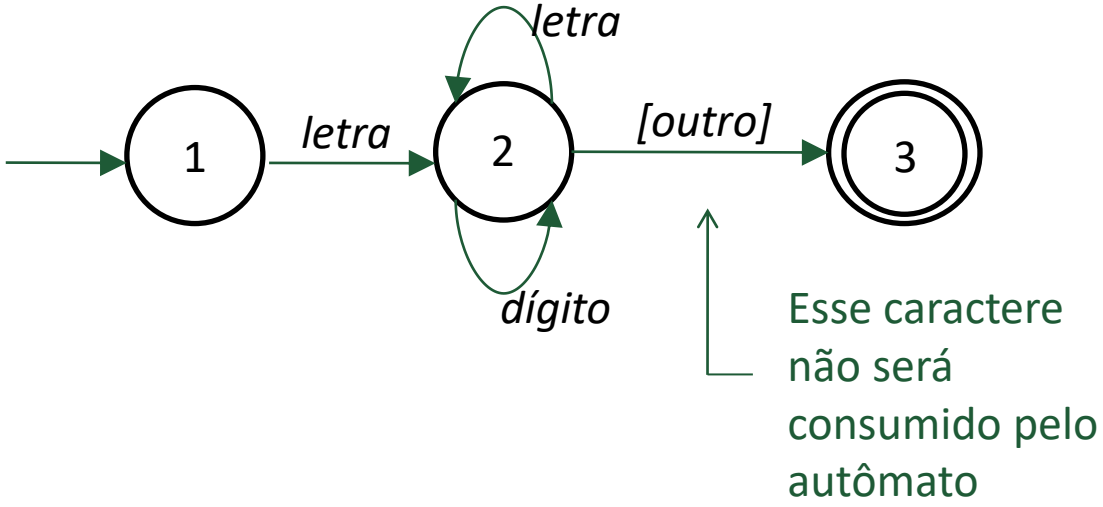
	s0	s2
0	C_1	C_1
1	C_2	C_2





De AFND para AFD e sua implementação

Há muitas formas de traduzir um AFD em código, como o seguinte AFD para reconhecimento de identificadores e palavras-chave

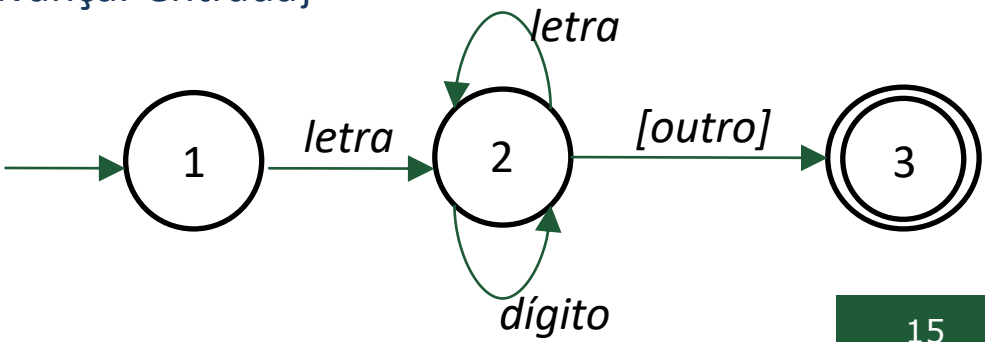


De AFND para AFD e sua implementação

Podemos simular esse AFD escrevendo o código da seguinte forma:

```
{início – estado 1}
If próximo caractere for letra then
    avance entrada;
    {estado 2}
    while próximo caractere for letra ou dígito
            avance entrada; {permanece no estado 2}
    end while;
    {passa para o estado 3 sem avançar entrada}
    aceitação;
else
    {erro ou outros casos}
end if;
```

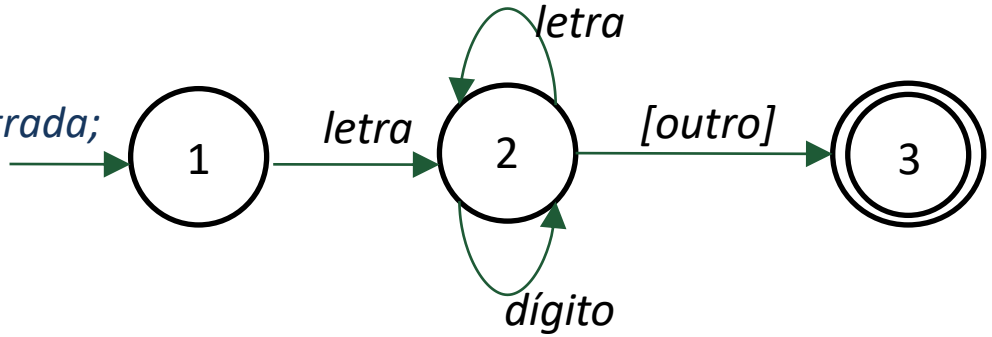
Indicado apenas para sistemas de varredura muito simples



De AFND para AFD e sua implementação

Uma alternativa melhor de implementação seria usar uma variável para manter o estado corrente e tratar as transições como declarações *case*

```
estado :=1; {início}
while estado = 1 ou 2
    case estado of
    1: case caractere de entrada of
                    letra:  avance entrada; estado := 2;
                    default estado := 4 {erro ou ignore};
          end case;
    2: case caractere de entrada of
                    letra, dígito: avance entrada;
                    default estado:=3;
          end case;
    end case;
end while;
If  estado = 3 then  aceitação else erro ou ignore;
```



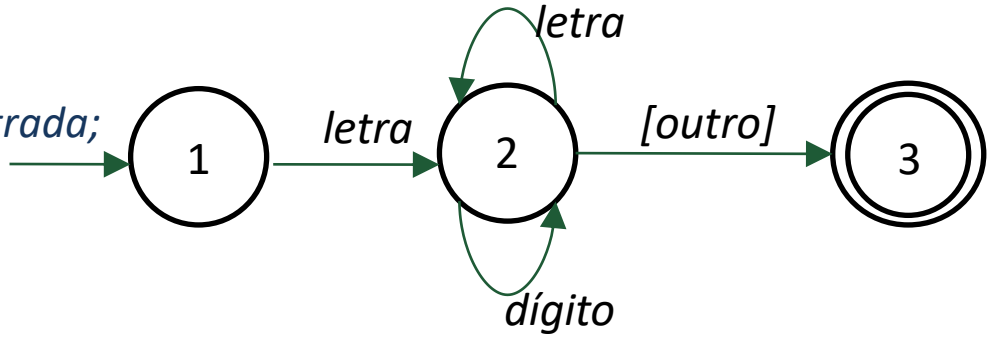
De AFND para AFD e sua implementação

Uma alternativa melhor de implementação seria usar uma variável para manter o estado corrente e tratar as transições como declarações *case*

```
estado :=1; {início}
while estado = 1 ou 2
    case estado of
    1: case caractere de entrada of
                    letra:  avance entrada; estado := 2;
                    default estado := 4 {erro ou ignore};
          end case;
    2: case caractere de entrada of
                    letra, dígito: avance entrada;
                    default estado:=3;
          end case;
    end case;
end while;
If  estado = 3 then  aceitação else erro ou ignore;
```

x=0

	x	=	0
Estado = 1			



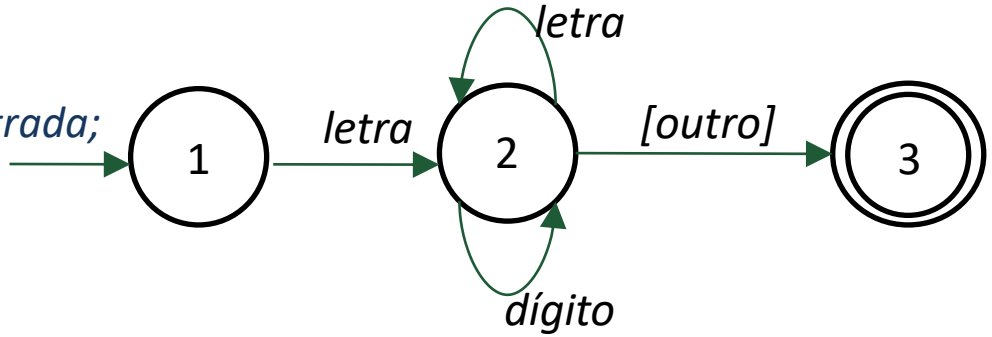
De AFND para AFD e sua implementação

Uma alternativa melhor de implementação seria usar uma variável para manter o estado corrente e tratar as transições como declarações *case*

```
estado :=1; {início}
while estado = 1 ou 2
    case estado of
    1: case caractere de entrada of
                    letra:  avance entrada; estado := 2;
                    default estado := 4 {erro ou ignore};
        end case;
    2: case caractere de entrada of
                    letra, dígito: avance entrada;
                    default estado:=3;
        end case;
    end case;
end while;
If  estado = 3 then  aceitação else erro ou ignore;
```

x=0

	x	=	0
Estado = 1	Estado = 2		



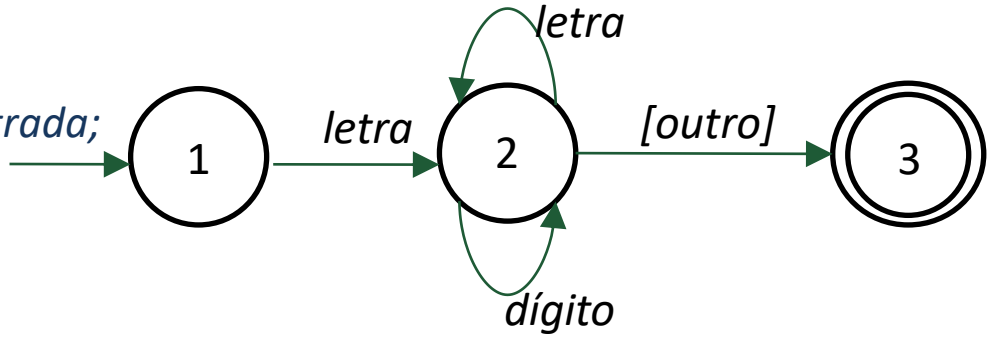
De AFND para AFD e sua implementação

Uma alternativa melhor de implementação seria usar uma variável para manter o estado corrente e tratar as transições como declarações *case*

x=0

	x	=	0
Estado = 1	Estado = 2	Estado = 3	

estado :=1; {início}
while *estado = 1 ou 2*
case *estado* **of**
1: case *caractere de entrada* of
letra: avance entrada; estado := 2;
default *estado := 4* {erro ou ignore};
end case;
2: case caractere de entrada of
letra, dígito: avance entrada;
default estado:=3;
end case;
end case;
end while;
If *estado = 3* **then** *aceitação* **else** *erro ou ignore;*



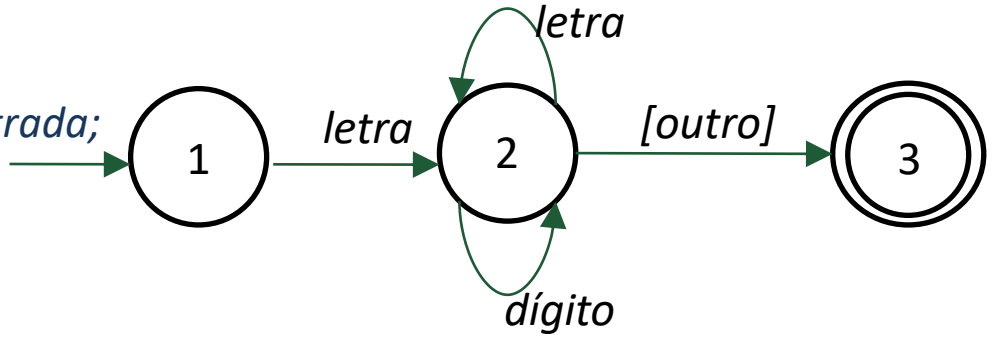
De AFND para AFD e sua implementação

Uma alternativa melhor de implementação seria usar uma variável para manter o estado corrente e tratar as transições como declarações *case*

x=0

	x	=	0
Estado = 1	Estado = 2	Estado = 3	

```
estado :=1; {início}
while estado = 1 ou 2
    case estado of
    1: case caractere de entrada of
                    letra:  avance entrada; estado := 2;
                    default estado := 4 {erro ou ignore};
          end case;
    2: case caractere de entrada of
                    letra, dígito: avance entrada;
                    default estado:=3;
          end case;
    end case;
end while;
If  estado = 3 then  aceitação else erro ou ignore;
```

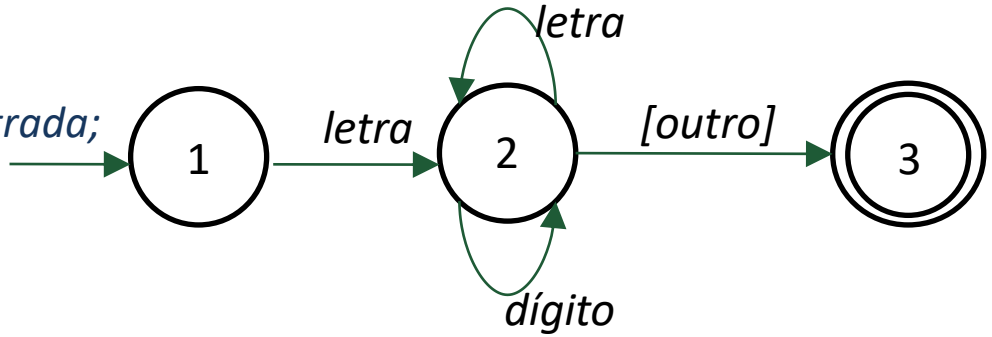


De AFND para AFD e sua implementação

Uma alternativa melhor de implementação seria usar uma variável para manter o estado corrente e tratar as transições como declarações *case*

```
estado :=1; {início}
while estado = 1 ou 2
    case estado of
    1: case caractere de entrada of
                    letra:  avance entrada; estado := 2;
                    default estado := 4 {erro ou ignore};
          end case;
    2: case caractere de entrada of
                    letra, dígito: avance entrada;
                    default estado:=3;
          end case;
    end case;
end while;
If  estado = 3 then  aceitação else erro ou ignore;
```

nota01=7;
nota02=6;



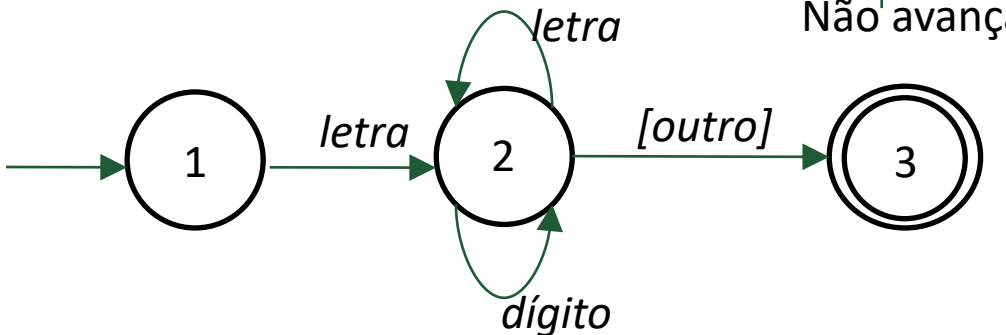
De AFND para AFD e sua implementação

- Algoritmos dirigidos por tabela proporcionam a implementação de um código genérico para AFD;
- Considere a tabela a seguir para o AFD anterior

caracteres de entrada

estados	letra	dígito	outro	Aceitação
1	2			não
2	2	2	[3]	não
3				sim

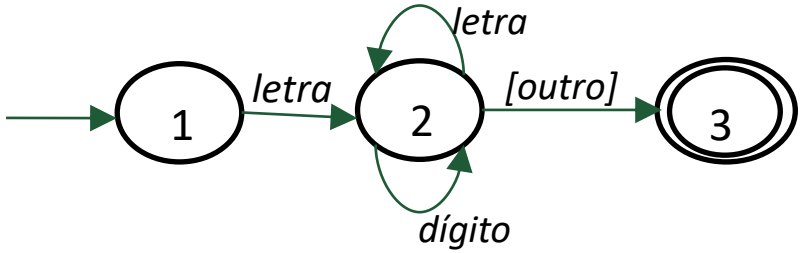
Não avança a entrada



De AFND para AFD e sua implementação

Um algoritmo dirigido por tabela para o AFD anterior poderia utilizar as seguintes matrizes:

- Uma matriz de inteiros *T[estado,ch]*
- Uma matriz booleana *Avance[estado,ch]*
- Uma matriz booleana *Aceita[estado]*




```
estado :=1;
erro:=F;
ch:=próximo caractere de entrada;
if ch not letra then erro:=V;
while not Aceita[estado] and not erro
    novoestado:= T[estado,ch];
    if Avance[estado,ch] then ch:=próximo caractere de entrada;
    estado:= novoestado;
end while;
if Aceita[estado] then aceitação;
```

	letra	dígito	outro	Aceitação
1	2 V			F
2	2 V	2 V	3 F	F
3				V

Exemplos: x=0; n1=2; n2=3;

De AFND para AFD e sua implementação

- Cabe ao sistema de varredura (analisador léxico) ler os caracteres do código-fonte e organizá-los em unidades lógicas para as outras partes do compilador (como o analisador sintático);
- É necessário implementar, como parte do sistema de varredura, uma rotina que leia os caracteres do arquivo de entrada (programa fonte), e retorne o caractere lido.

De AFND para AFD e sua implementação

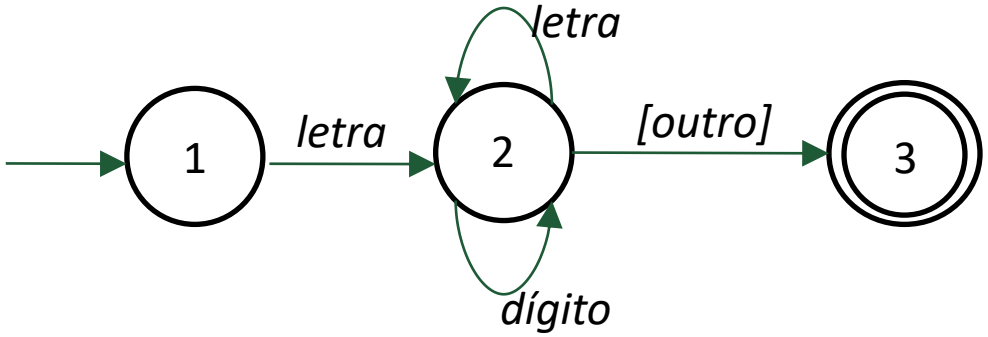
Como alternativa, considere a função *proximoChar()* apresentada no código a seguir:

```
#include <fstream>
#include <iostream>
using namespace std;

ifstream arq("teste.txt");

char proximoChar()
{
  char ch;
  arq.get(ch);  // lê um caractere do arquivo texto
  return(ch);
}

int main()
{
  char ch;
  while(!arq.eof())
  {
   ch = proximoChar();
   cout << ch;
  }
  return 0;
}
```

De AFND para AFD e sua implementação

Atividade

Escreva um programa (em C ou C++) que leia um arquivo de texto e implemente o algoritmo dirigido por tabela para reconhecer identificadores e palavras-chave (conforme AFD abaixo). Gerar um programa de saída, substituindo todos os identificadores pelo token **ID**.

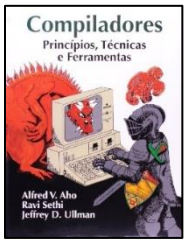


letra = [a-zA-Z]
dígito = [0-9]
ID = letra(letra|dígito)*

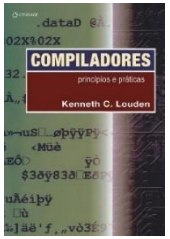
De AFND para AFD e sua implementação

Bibliografia consultada

RICARTE, I. **Introdução à Compilação.** Rio de Janeiro: Editora Campus/Elsevier, 2008.



AHO, A. V.; LAM, M. S.; SETHI, R. e ULLMAN, J. D. **Compiladores: princípios, técnicas e ferramentas.** 2ª edição – São Paulo: Pearson Addison-Wesley, 2008.



LOUDEN, K. C. **Compiladores: princípios e práticas.** São Paulo: Pioneira Thompson Learning, 2004.

Agradeço ao Professor Luiz Eduardo G. Martins pelo material disponibilizado.